Data Analysis in Geophysics
ESCI 7205

Bob Smalley
Room 103 in 3892 (long building), x-4929

Tu/Th - 13:00-14:30
CERI MAC (or STUDENT) LAB

Lab – 3, 09/03/13

More on vectorization.

MATLAB is a vectorized high level language

Requires change in programming style
(if one already knows a non-vectorized programming language such as Fortran, C, Pascal, Basic, etc.)

Vectorized languages allow operations over arrays using simple syntax, essentially the same syntax one would use to operate over scalars.
(looks like math again.)

What is vectorization?
(with respect to matlab)

Vectorization is the process of writing code for MATLAB that uses matrix operations or other fast built-in functions instead of using explicit loops.

The benefits of doing this are usually sizeable.

The reason for this is that MATLAB is an <u>interpreted</u> language. Function calls have very high overhead, and indexing operations (inherent in a loop operation) are not particularly fast.

Loop versus vectorized version of same code. New commands "**tic**" and "**toc**" - time the execution of the code between them.

```
>> a=rand(1000);
>> tic;b=a*a;toc
Elapsed time is 0.229464 seconds.
>> tic;for k=1:1000,for l=1:1000,c(k,l)=0;for m=1:1000, c(k,l)=c
(k,l)+a(k,m)*a(m,l);end, end, end, toc
Elapsed time is 22.369451 seconds.
>> whos
  Name         Size              Bytes  Class     Attributes
  a            1000x1000       8000000  double
  b            1000x1000       8000000  double
  c            1000x1000       8000000  double
  k               1x1                8  double
  l               1x1                8  double
  m               1x1                8  double

>> max(max(b-c))
ans =
   9.6634e-13
```

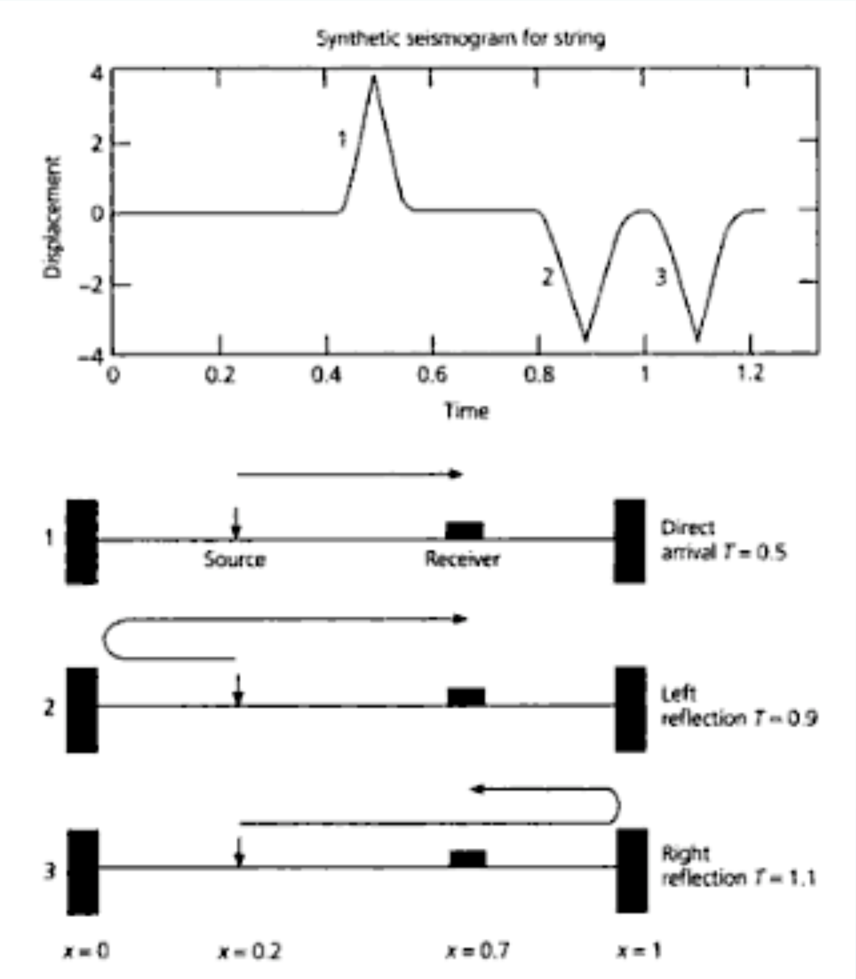
Factor 100 difference in time for multiplication of 10^3x10^3 matrix!

Vectorization of synthetic seismogram example from

Stein and Wysession

Intro to Seismology and Earth Structure.

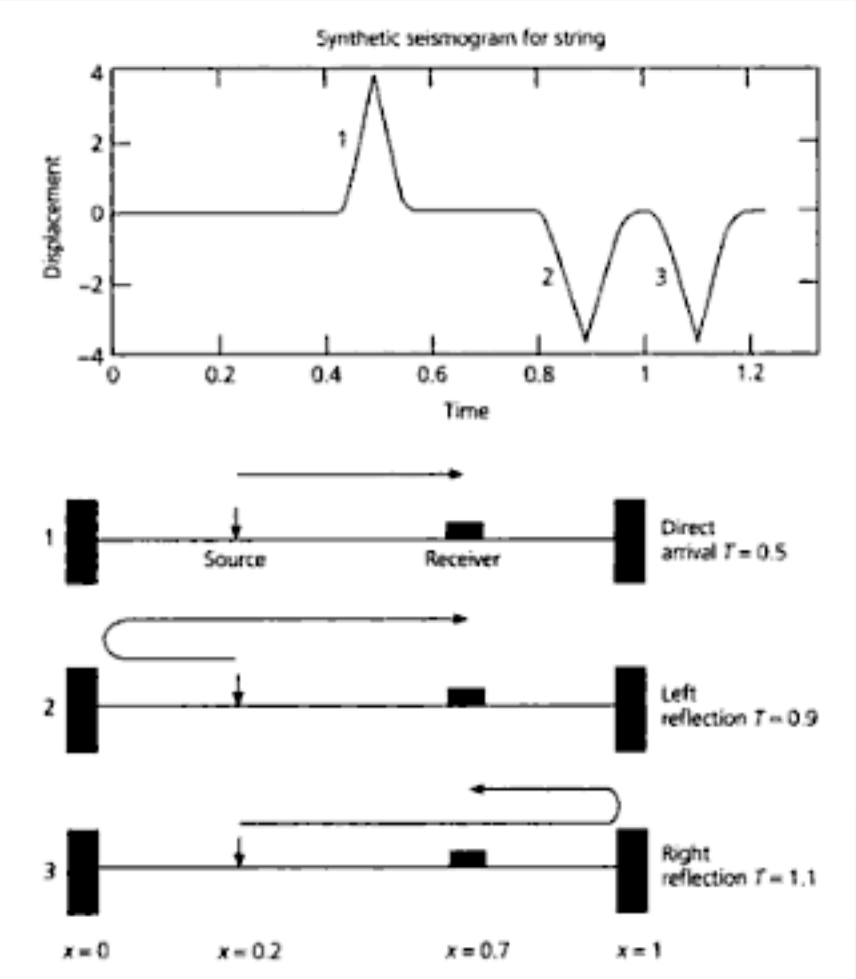
Section on scientific programming



$$u(x,t) = \sum_{n=1}^{\infty} \sin\left(n\pi x / L\right) \sin\left(n\pi x_s / L\right) \cos\left(\omega_n t\right) \exp\left[-\left(\omega_n \tau\right)^2 / 4\right]$$

Start by just "translating" the Fortran code into Matlab.

So far we probably don't fully understand the math, but we have a formula and so we can calculate u(x,t).



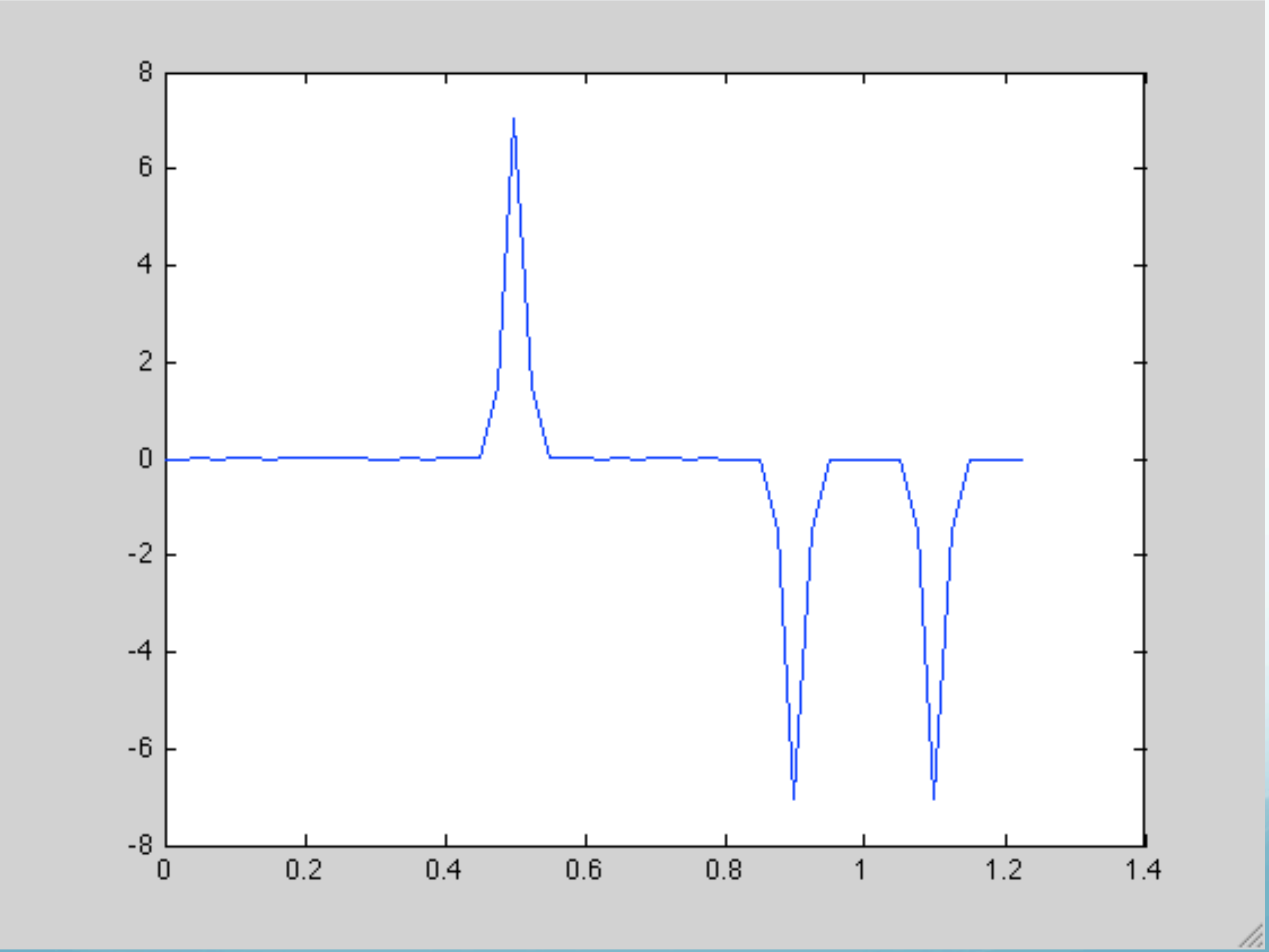
$$u(x,t)=\sum_{n=1}^{\infty}\sin\left(n\pi x/L\right)\sin\left(n\pi x_s/L\right)\cos\left(\omega_n t\right)\exp\left[-\left(\omega_n\tau\right)^2/4\right]$$

Doing translation for homework

```
%synthetic seismogram for homogeneous
string, u(t)
%calculated by normal mode summation
%string length
alngth=1;
%velocity m/sec
c=1.0;
%number modes
nmode=200;
%source position
xsrc=0.2;
%receiver position
xrcvr=0.7;
%seismogram time duration
tdurat=1.25;
%number time steps
nstep=50;
%time step
dt=tdurat/nstep;
%source shape term
tau=0.02;
fprintf('%s\n','synthetic seismogram for
string')
fprintf('%s %0.5g\n','number modes',
nmode)
fprintf('%s %0.5g %0.5g\n','length and
velocity', alngth, c)
fprintf('%s %0.5g %0.5g\n','posn src and
rcvr',xsrc,xrcvr)
fprintf('%s %0.5g %0.5g %0.5g\n','durn,
time steps, del t',tdurat,nstep,dt)
fprintf('%s %0.5g\n','source shape',
tau)
%initialize displacement
for cnt=1:nstep
    u(cnt)=0;
end
for k=1:nstep
    t(k)=dt*(k-1);
end
%outer loop over modes
for n=1:nmode
    anpial=n*pi/alngth;
%space terms - src & rcvr
    sxs=sin(anpial*xsrc);
    sxr=sin(anpial*xrcvr);
%mode freq
    wn=n*pi*c/alngth;
%time indep terms
    dmp=(tau*wn)^2;
    scale=exp(-dmp/4);
    space=sxs*sxr*scale;
%inner loop oner time  steps
    for k=1:nstep
%           t=dt*(k-1);
%           cwt=cos(wn*t);
        cwt=cos(wn*t(k));
%compute disp
        u(k)=u(k)+cwt*space;
    end
end
plot(t,u)
```

Slightly cleaned up version of Fortran program in Stein and Wysession "translated" to Matlab. (took calculation of time out of inner loop – is recalculated each time through, waste of time, calculate as vector once at beginning)

Synthetic seismogram produced by Matlab code translated from Fortran.

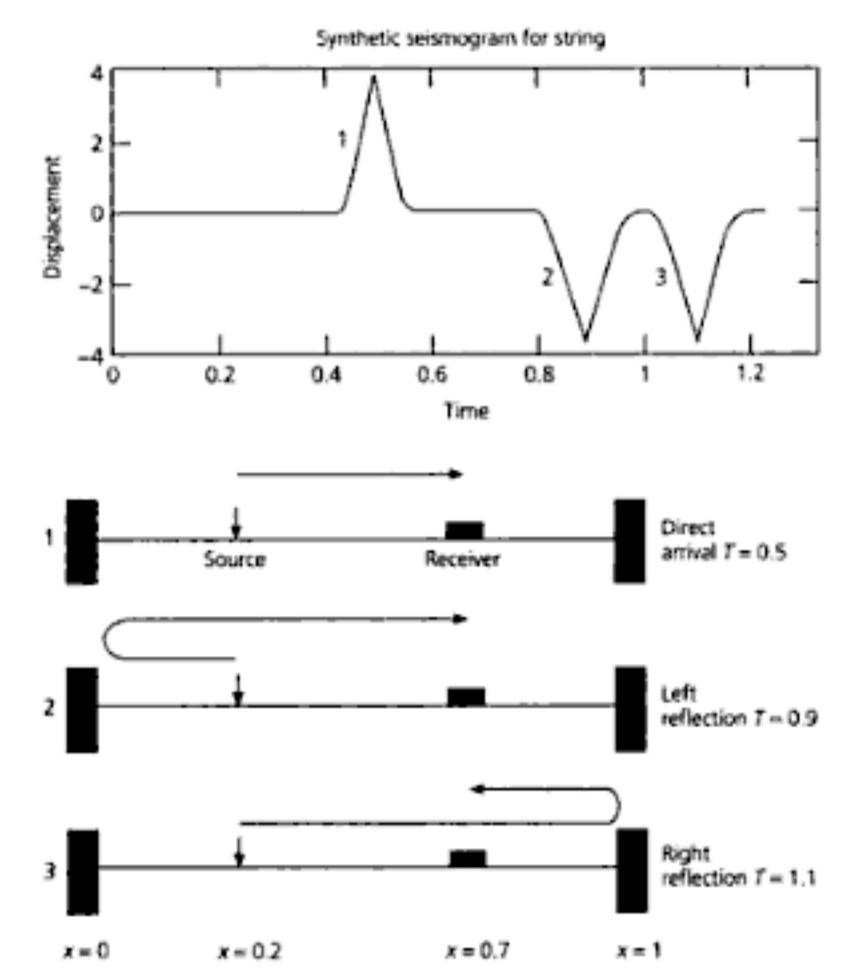


Variables

```
>> whos
  Name         Size            Bytes  Class     Attributes
  alngth       1x1                 8  double
  anpial       1x1                 8  double
  c            1x1                 8  double
  cnt          1x1                 8  double
  cwt          1x1                 8  double
  dmp          1x1                 8  double
  dt           1x1                 8  double
  k            1x1                 8  double
  n            1x1                 8  double
  nmode        1x1                 8  double
  nstep        1x1                 8  double
  scale        1x1                 8  double
  space        1x1                 8  double
  sxr          1x1                 8  double
  sxs          1x1                 8  double
  t            1x1                 8  double
  tau          1x1                 8  double
  tdurat       1x1                 8  double
  u            1x50              400  double
  wn           1x1                 8  double
  xrcvr        1x1                 8  double
  xsrc         1x1                 8  double
```

Let's return to the original problem and try to understand what is going on.

We will use this to understanding to further vectorize (speed up) the code.



$$u(x,t)=\sum_{n=1}^{\infty}\sin\left(n\pi x/L\right)\sin\left(n\pi x_s/L\right)\cos\left(\omega_n t\right)\exp\left[-\left(\omega_n\tau\right)^2/4\right]$$

This is just the Fourier domain representation for waves on a string with fixed ends

$$u(x,t)=\sum_{n=1}^{\infty}\sin\left(n\pi x_s/L\right)\sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)\exp\left[-\left(\omega_n\tau\right)^2/4\right]$$

$$\left(Note:\ \ \omega_n=n*\omega_0\right)$$

$$u(x,t)=\sum_{n=1}^{\infty}\boxed{\left(\sin\left(n\pi x_s/L\right)\exp\left[-\left(\omega_n\tau\right)^2/4\right]\right)}\ \sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)$$

Weight - no dependence on x or t, only ω_n.

$$u(x,t)=\sum_{n=1}^{\infty}\boxed{a_n}\boxed{\sin\left(n\pi x/L\right)\cos\left(\omega_n t\right)}$$

Standing wave made from 2 opposite direction traveling waves. Amplitude varies with time, but does not "move"

$$u(x,t)=\sum_{n=1}^{\infty}a'_n\boxed{\left[\cos\left(n\pi x/L+\omega_n t\right)+\cos\left(n\pi x/L-\omega_n t\right)\right]}$$

Going left Going right

Normal Mode (and combination of traveling waves to make standing wave) formulation for displacement of a string

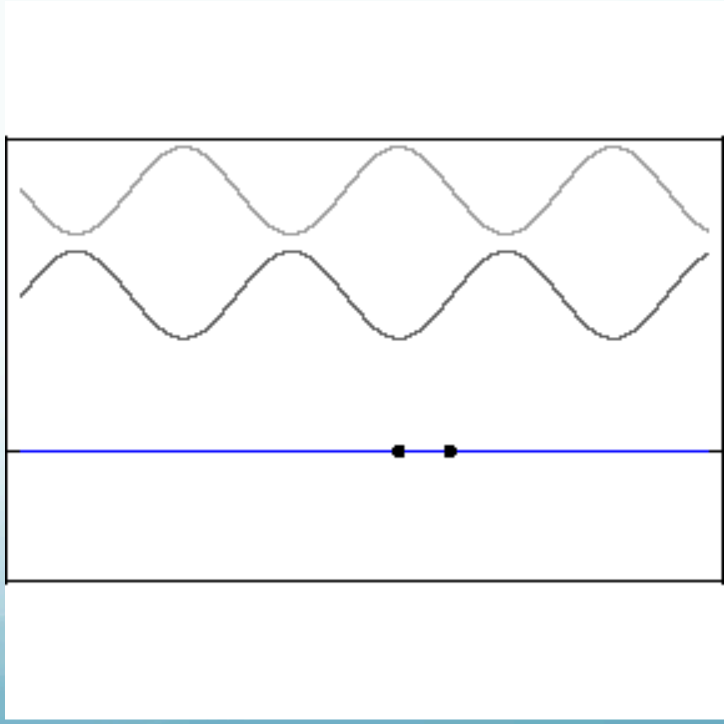
$$u(x,t) = A\cos\ (kx + \omega t) + A\cos(kx - \omega t)$$

$$u(k,\omega) = A\cos\ (kx + \omega t) + A\cos(kx - \omega t)$$

$$u(x,t) = u(k,\omega) = 2A\cos(\omega t)\cos(kx)$$

$$u_n(x,t) = \cos(k_n x / L)\cos(\omega_n t)$$

$$where \qquad \omega_n = v\ k_n$$



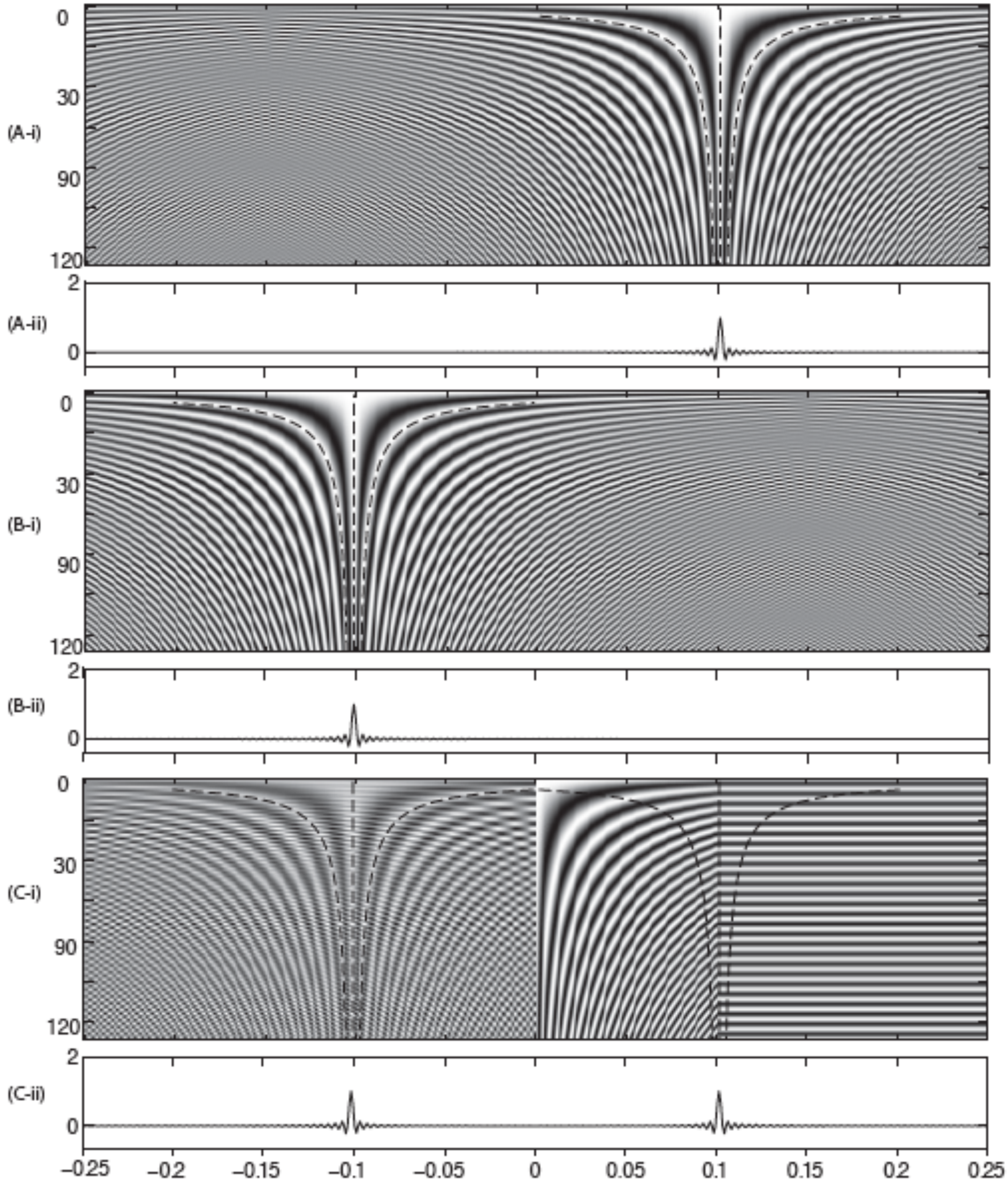
This is a sinusoidal wave that is fixed in space, cos(kx), whose amplitude is modulated harmonically in time, cos (ωt)

Do over a range of frequencies.

Delta functions going right on top

and left in middle

and combined (bottom).



u(x,t)=Acos (kx+ωt)+Acos(kx−ωt)
u(k,ω)=Acos (kx+ωt)+Acos(kx−ωt)

u(x,t)=u(k,ω)=2Acos(ωt)cos(kx)

Look at the basic element of Fourier series, weighted sum of sin and cos functions

(look at cos only to see how works).

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m)$$

$$u(t_m) = \frac{a_0}{2} + \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \end{pmatrix} \bullet \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \end{pmatrix} \Longleftarrow \text{Dot product}$$

or matrix multiply: ac=(c'a')'

$$u(t_m) = \frac{a_0}{2} + \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \end{pmatrix} \begin{pmatrix} \cos(\omega_1 t_m) \\ \cos(\omega_2 t_m) \\ \cos(\omega_3 t_m) \\ \cdots \\ \cos(\omega_n t_m) \end{pmatrix} = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \boxed{\cos(\omega_1 t_m)} & \boxed{\cos(\omega_2 t_m)} & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\ \vec{a}$$

matrix multiply, at multiple times to make full seismogram

Look at the basic Fourier series

At constant time, weighted sum of cosines at different frequencies at that time

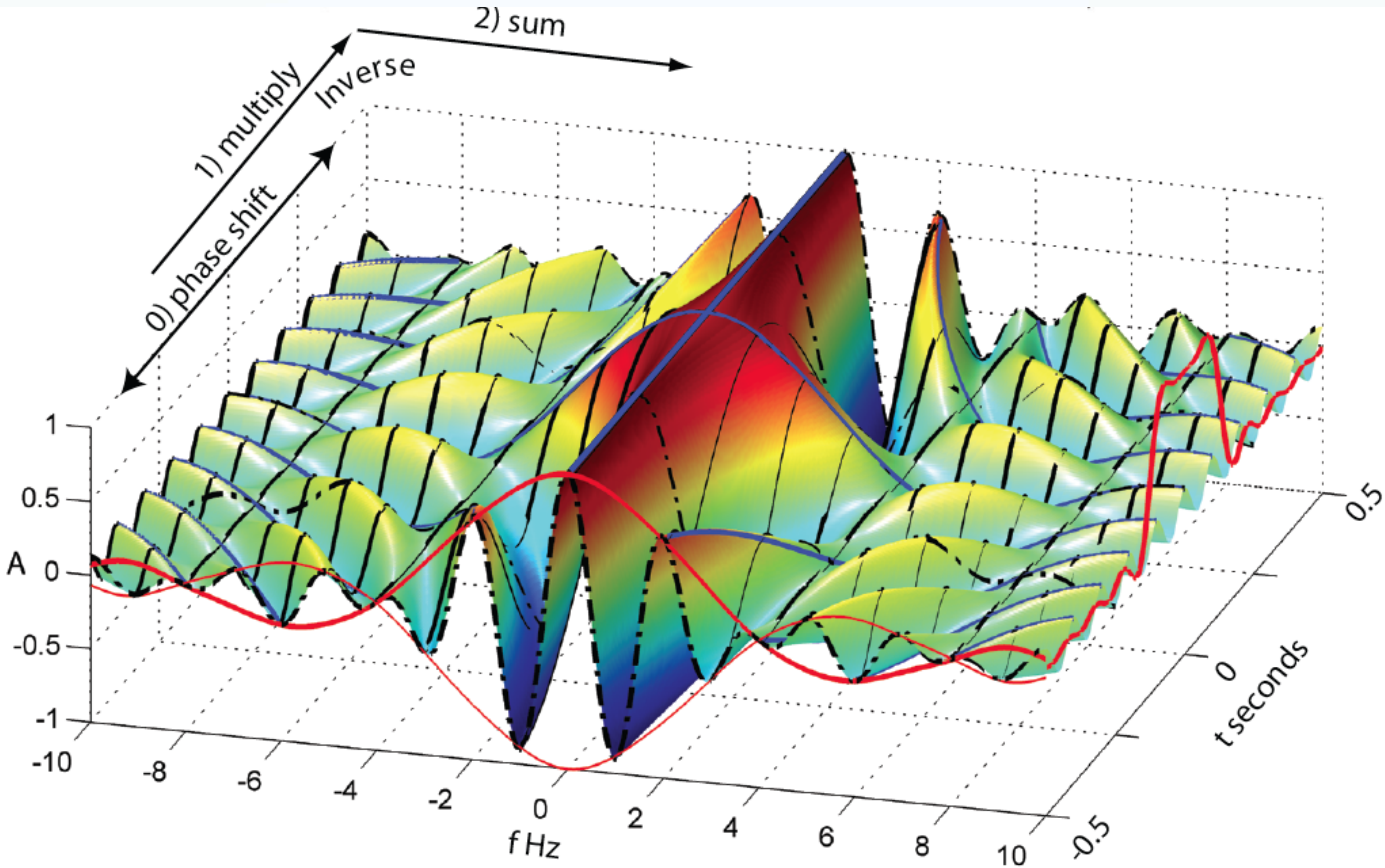
$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\,\vec{a}$$

constant frequency cosine as function of time (basis functions)

This is multiplication of a matrix (with cosines as functions of frequency – across - and time - down) times a vector containing the Fourier series weights.

We have just vectorized the equations for the Fourier series!



Even though this is a major improvement over doing this with for loops, and is clear conceptually, it is still not "computable" as it takes $O(N^2)$ operations (and therefore time) to do it. This is OK for small `N`, but quickly gets out of hand.

Fourier analysis is typically done using the Fast Fourier transform (FFT) algorithm – which has $O(N \log_2 N)$ operations and is significantly faster for large `N`.

Fourier decomposition.

"Basis" functions are the sine and cosine functions.

Notice that first sine term is all zeros (so don't really need it) and last sine term (not shown) is same as last cosine term, just shifted one – so will only need one of these).

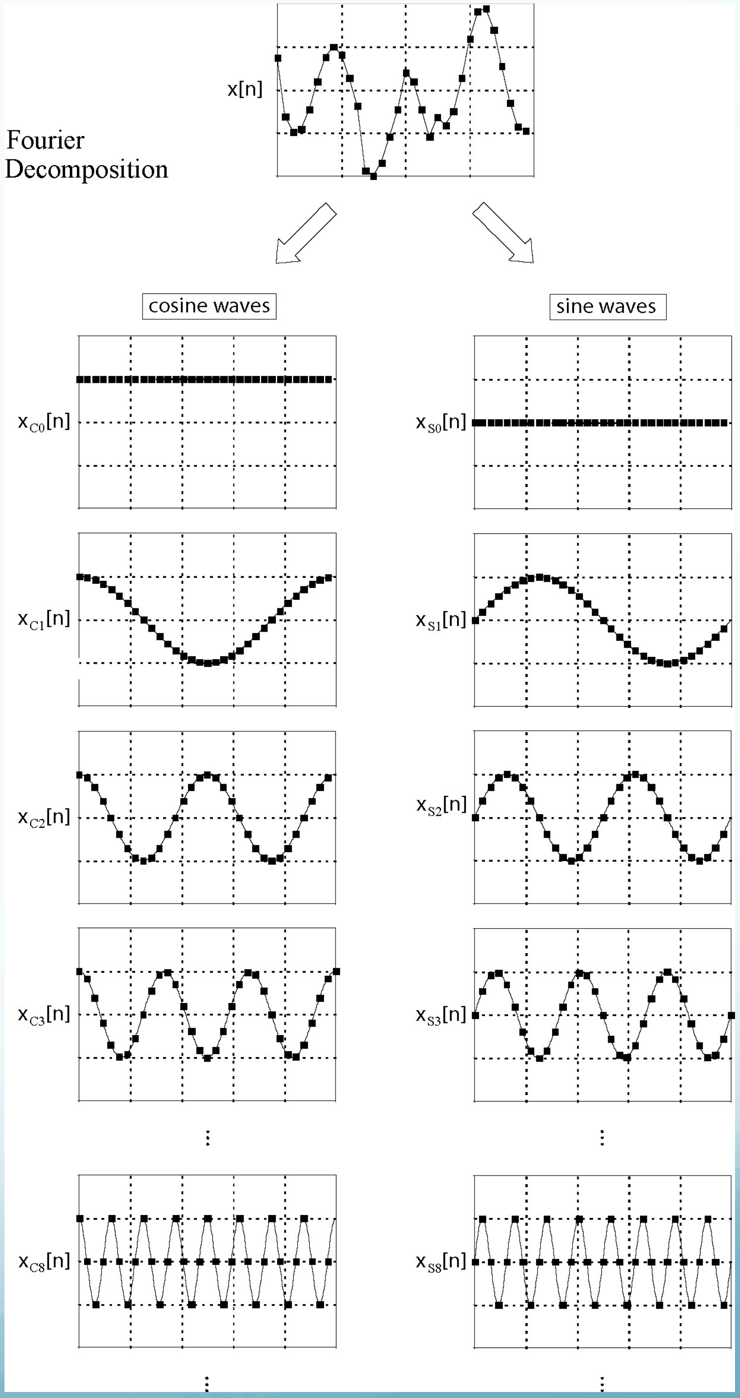


Figure from Smith

Fourier transform (actually Fourier series)

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m) + \sum_{n=1}^{N} b_n \sin(\omega_n t_m)$$

The Fast Fourier Transform (FFT) depends on noticing that there is a lot of repetition in the calculations – each higher frequency basis function can be made by selecting points from the ω_0 function. The weight value is multiplied by the same basis function value an increasing number of times as ω increases.

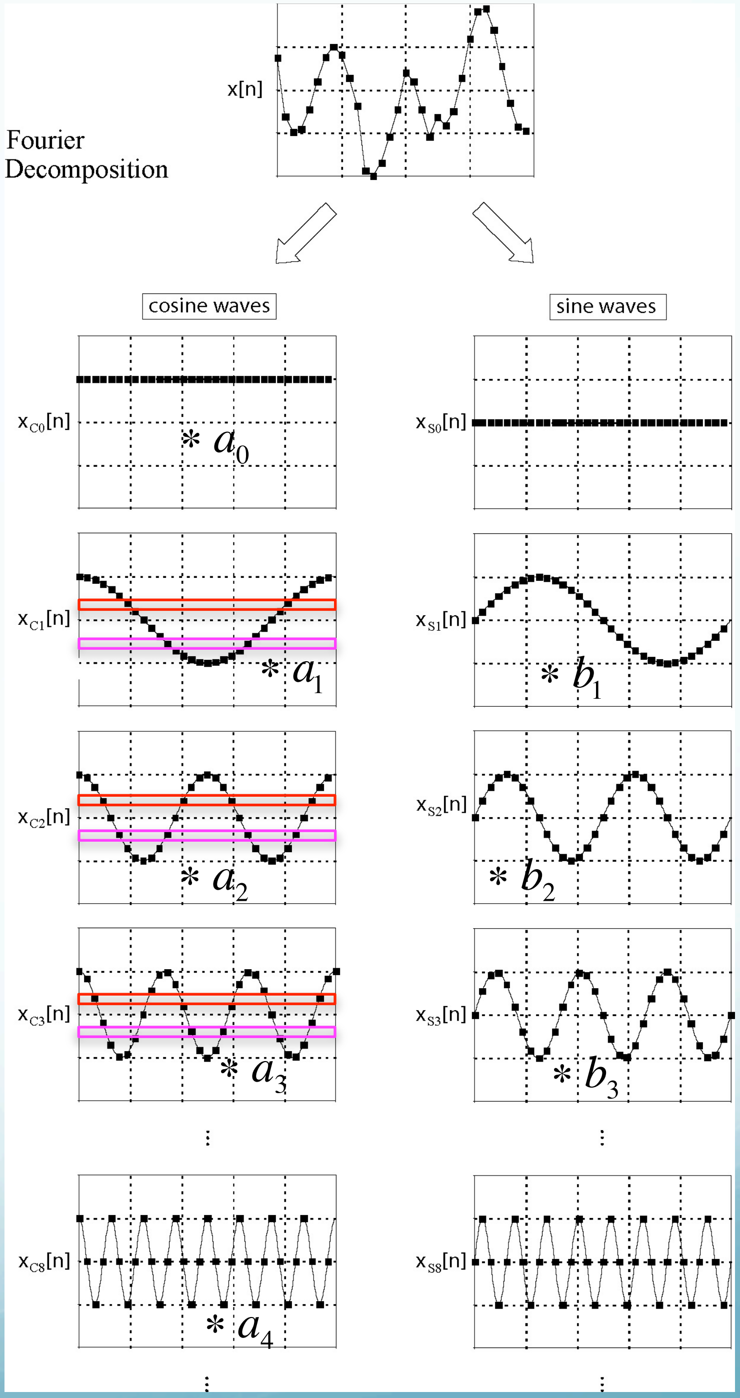


Figure from Smith

FFT

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m) + \sum_{n=1}^{N} b_n \sin(\omega_n t_m)$$

The FFT uses regularities of the calculation to calculate the basis functions and then basically does each unique multiplication only once, stores it, and then does the bookeeping to add them all up correctly.

The points in the trace at the top are made from vertical sums of the weighted points at the same time in the `cos` and `sin` traces in the bottom.

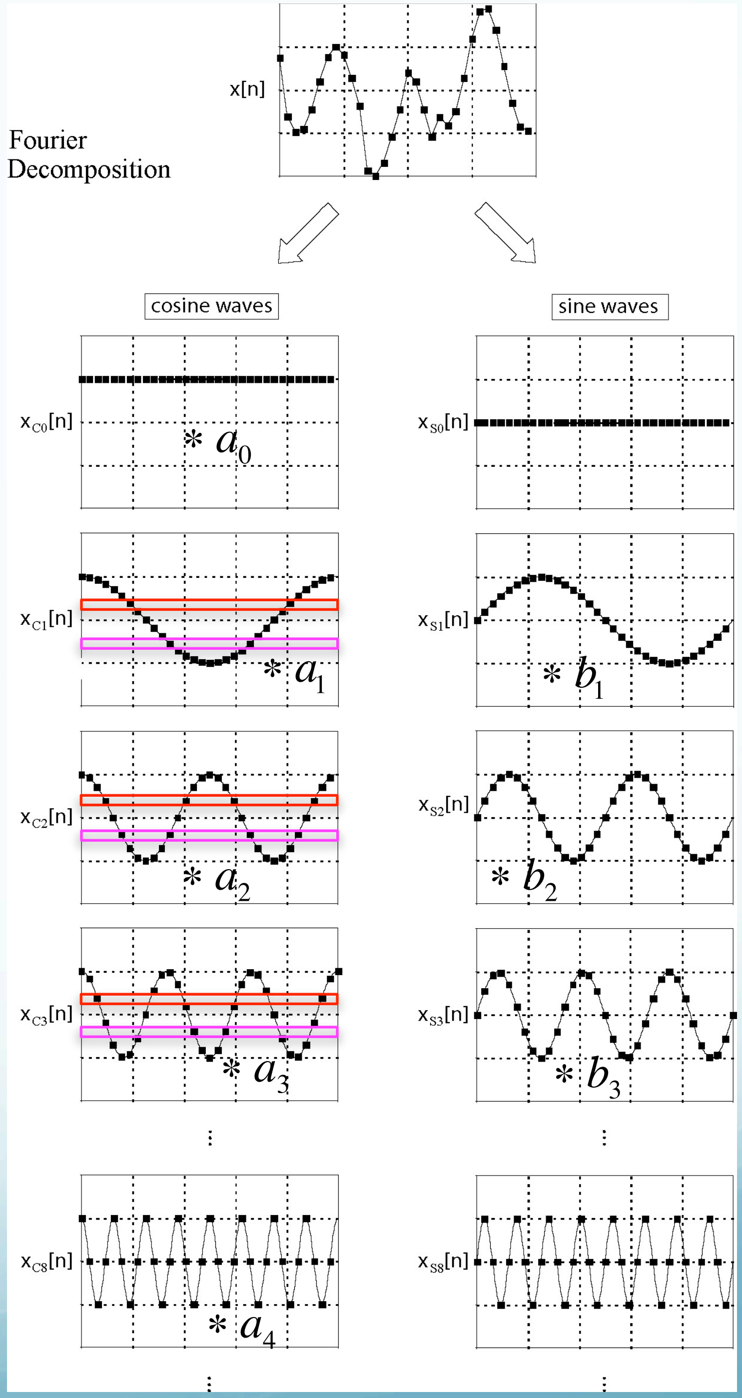


Figure from Smith

FFT

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(\omega_n t_m) + \sum_{n=1}^{N} b_n \sin(\omega_n t_m)$$

$$u(t_m) = \frac{a_0}{2} + \sum_{n=1}^{N} c_n W_N^{mn}, \qquad W = e^{-i2\pi/N}$$

The FFT uses the following symmetry properties

Symmetry $$W_N^{k+N/2} = -W_N^k$$

Periodicity $$W_N^{k+N} = W_N^k$$

FFT needs number points = power of 2.

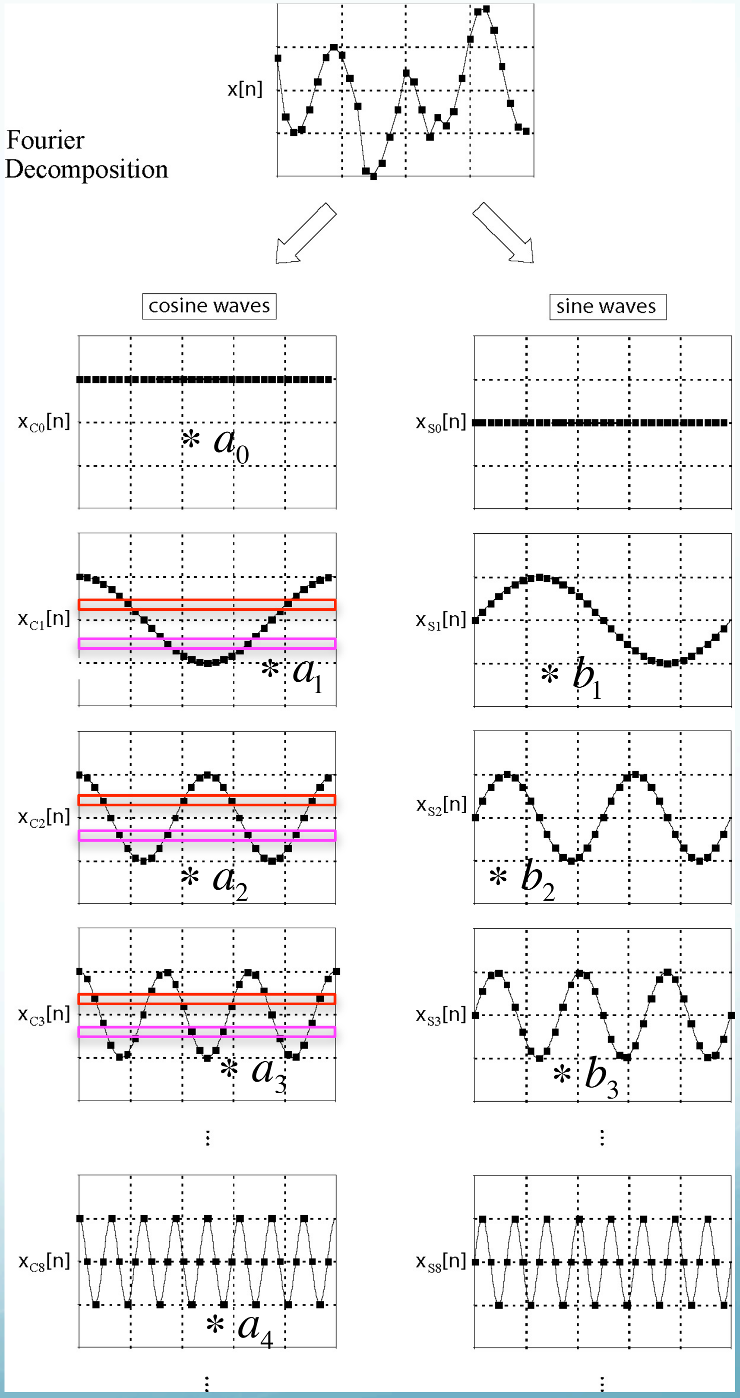


Figure from Smith

```
% number of time samples M
% points
% source/receiver position:
% xs/xr (meters)
% speed c (meters/sec)
% length L (meters)
% number of modes N
% source pulse duration Tau
% (sec)
% length of seismogram T (sec)

M=50;
xs=0.25;
xr = 0.7;
c=1;
L=1;
N=200;
Tau=0.02;
T=1.25;

%time vector, 1 row by M
%columns: start, step, stop
%will need lots, calc once
dt=T/M;
t=0:dt:T-dt;
```

Same program in Matlab after vectorization (is mostly comments!)

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \begin{pmatrix} \cos(\omega_1 t_m) & \cos(\omega_2 t_m) & \cos(\omega_3 t_m) & \cdots & \cos(\omega_n t_m) \\ \cos(\omega_1 t_{m+1}) & \cos(\omega_2 t_{m+1}) & \cos(\omega_3 t_{m+1}) & \cdots & \cos(\omega_n t_{m+1}) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cos(\omega_1 t_{m+k}) & \cos(\omega_2 t_{m+k}) & \cos(\omega_3 t_{m+k}) & \cdots & \cos(\omega_n t_{m+k}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_n \end{pmatrix}$$

$$\vec{u}(t_m : t_{m+k}) = \frac{a_0}{2} + \vec{W}\,\vec{a}$$

$$\left(Note:\ \omega_n = n * \omega_0\right)$$

$$a_n = \frac{1}{2}\left(\sin\left(n\pi x_s / L\right)\exp\left[-\left(\omega_n \tau\right)^2 / 4\right]\right)\sin\left(n\pi x_r / L\right)$$

We need to make the matrix and the vector

Making the matrix.

What size does it have to be?

What does each row and column represent?

There are N=200 columns for the M frequencies

There are N=50 rows for the N samples in the seismogram time series.

How do we make the elements (k,l) of the matrix?

Use fact that values needed are proportional to k and l.

Make appropriate vectors for time and frequency.

How big is each?

How combine them to make the matrix as a function of k and l?

Multiply elements of the matrix by dt and ω_0.

Take cosine of matrix.

Now calculate the weights.

Note the weights depend on n, and ω_n, but not t.

All t dependence is in the matrix elements.

So now have matrix with the trigonometric basis functions and a vector of the weights.

Just multiply them!
(careful with sizes)

This is not the way it is typically done (although some people still do it the "Fortran" way) as it is still $O(N^2)$.

The Matlab matrix multiplication method is faster than the Matlab loop method.
(a good Fortran compiler will beat the pants off either implementation in Matalb).

We did it this way for educational purposes.

Typically, it is done using the FFT (Fast Fourier Transform) algorithm which avoids duplication of effort in the multiplications and results in $O(N \log_2 N)$ multiplications.

For a time series 2^{16}=65,536

(FFT needs number of points = power of two, this is pretty typical number of points in seismogram, about 10 minutes at 100 Hz sampling)

We need O(16 * 2^{16})=1,048,576

Vs

4,294,967,296

Multiplications (slow)

(ratio 2.44140625e-4 -> 4096 times faster!!)

Two lessons

Vecotorizing Matlab (turn loops into matrix operations) makes Matlab go lots faster.
Should do it.

Vectorizing in general

- is not algorithmic
- is case specific

can give gigantic speed improvements (much more than Matlab style vectorizing) and even make something that is non-computable, computable.

But is lots of work - an art!

Get same figure as before.